Berkeley
UNIVERSITY OF CALIFORNIA

BERKELEY LAB

U.S. DEPARTMENT OF ENERGY

Implementing Graphblas Primitives on Distributed-Memory Systems

SIAM CSE'21

Minisymposium on GraphBLAS

Benjamin Brock, Aydın Buluç, and Katherine Yelick

March 1, 2021

Implementing Graphblas Primitives on Distributed-Memory Systems

Using RDMA!

SIAM CSE'21

Minisymposium on GraphBLAS

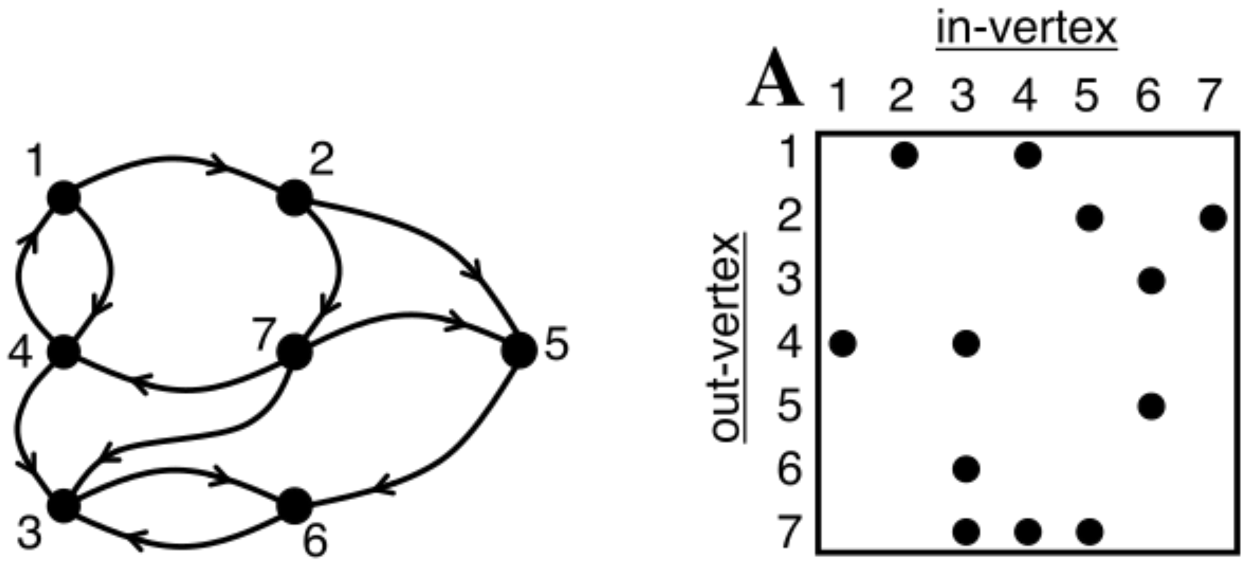
Benjamin Brock, Aydın Buluç, and Katherine Yelick

March 1, 2021

Background

Background

- **GraphBLAS API** allows graph algorithms to be expressed using **linear algebra primitives**

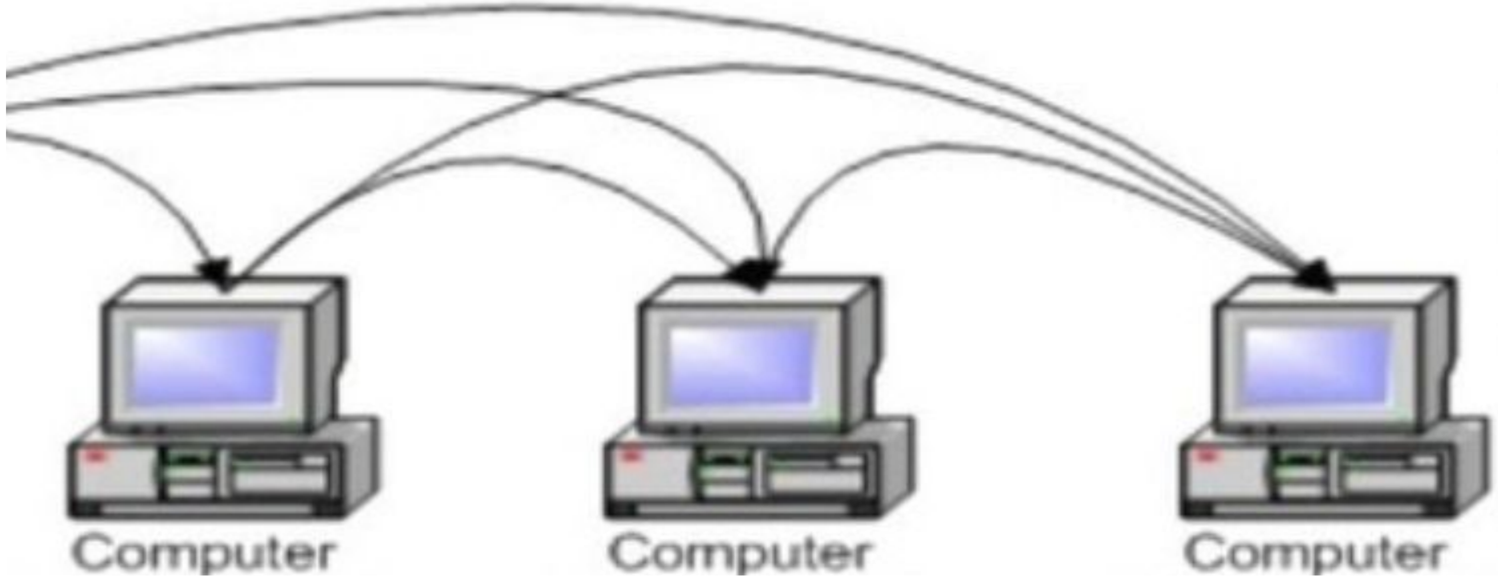


- Instead of optimizing **each graph algorithm individually**, **optimize only a few** sparse linear algebra operations

- Center around **matrix multiplication**: SpMM, SpGEMM, SpMV

What is “Distributed”?

- A collection of **nodes**, connected by a **network**.



How to program distributed?

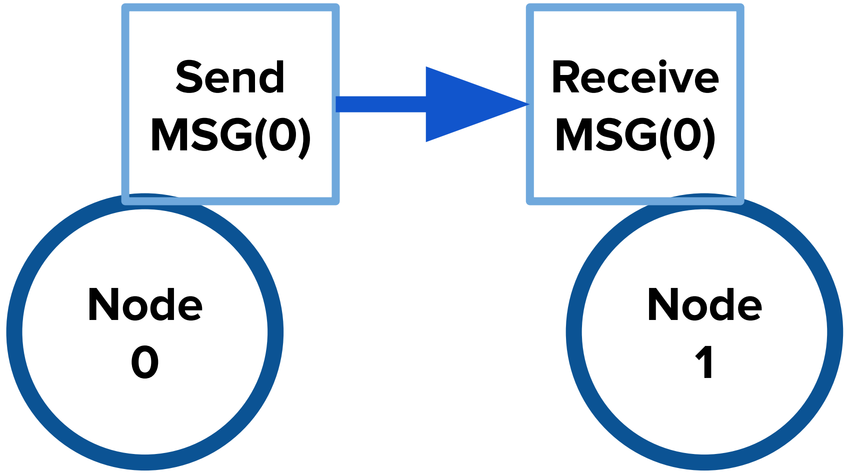
- **Message Passing** - **bulk synchronous** collectives (OR matching send/receives)

- **RDMA** - directly read/write to **remote memory**

How to program distributed?

- **Message Passing** - **bulk synchronous** collectives (OR matching send/receives)

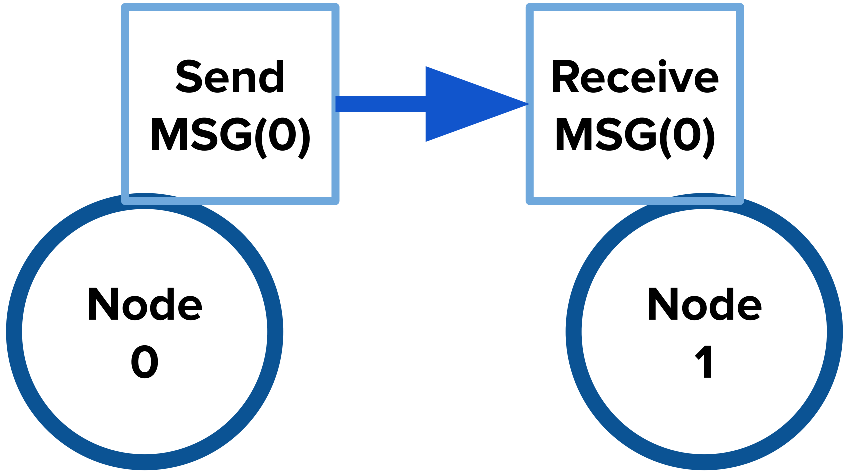
- **RDMA** - directly read/write to **remote memory**

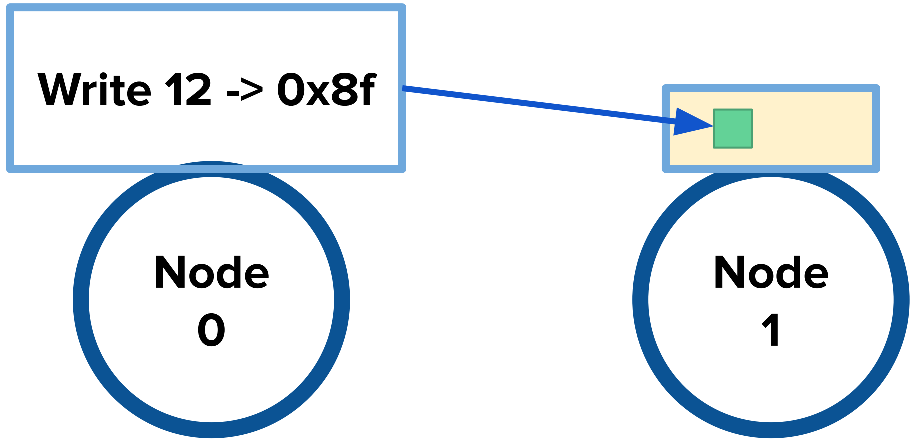


How to program distributed?

- **Message Passing - bulk synchronous** collectives (OR matching send/receives)

- **RDMA** - directly read/write to **remote memory**



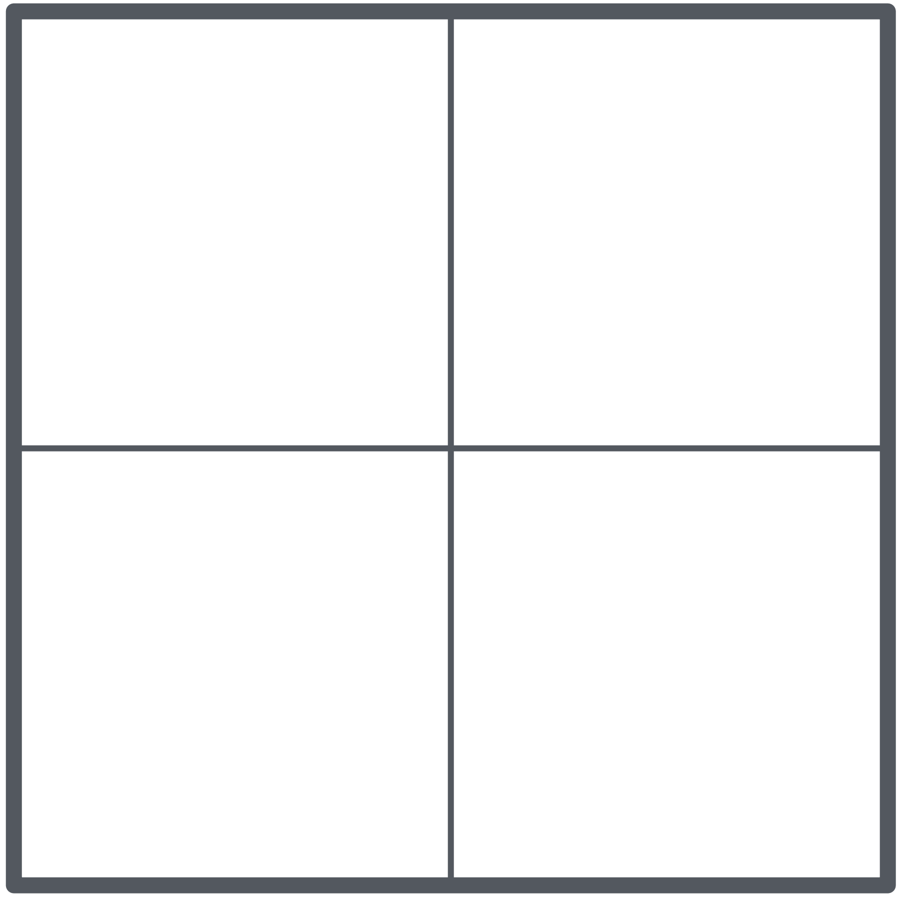


RDMA Availability Backup Slides

Distributed Matrices

Distributed Matrices

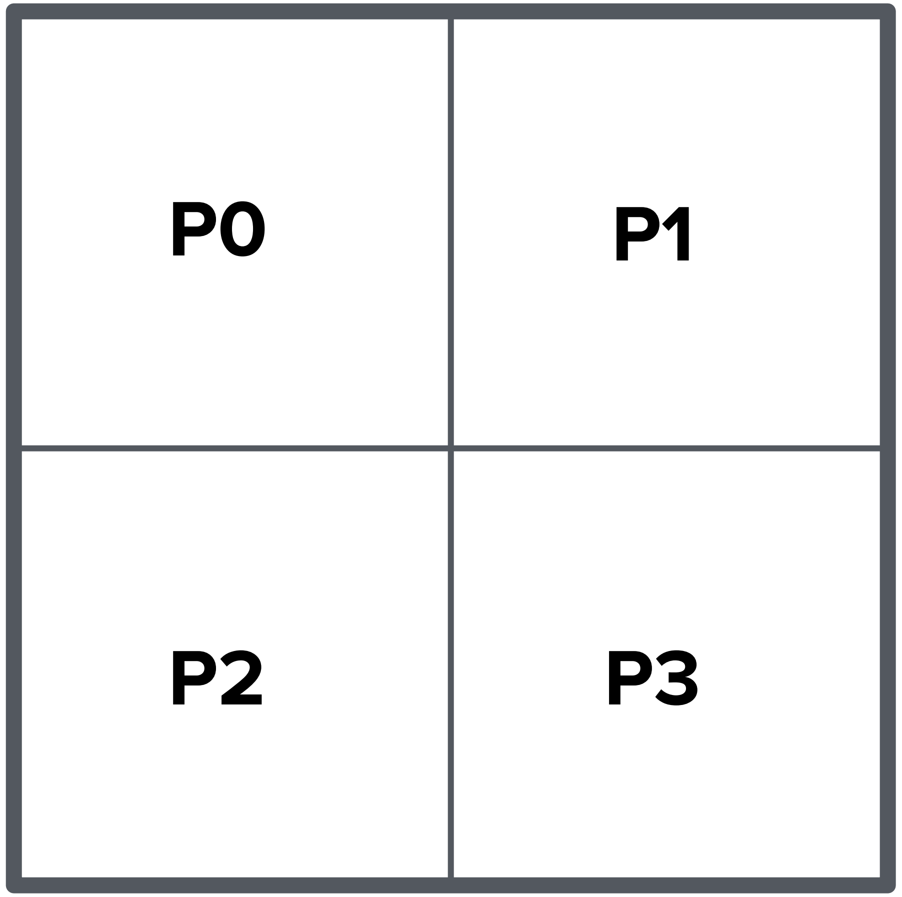
- Matrix is split up across a **tile grid** (composed of **tiles** or **blocks**)



Distributed Matrices

- Matrix is split up across a **tile grid** (composed of **tiles** or **blocks**)

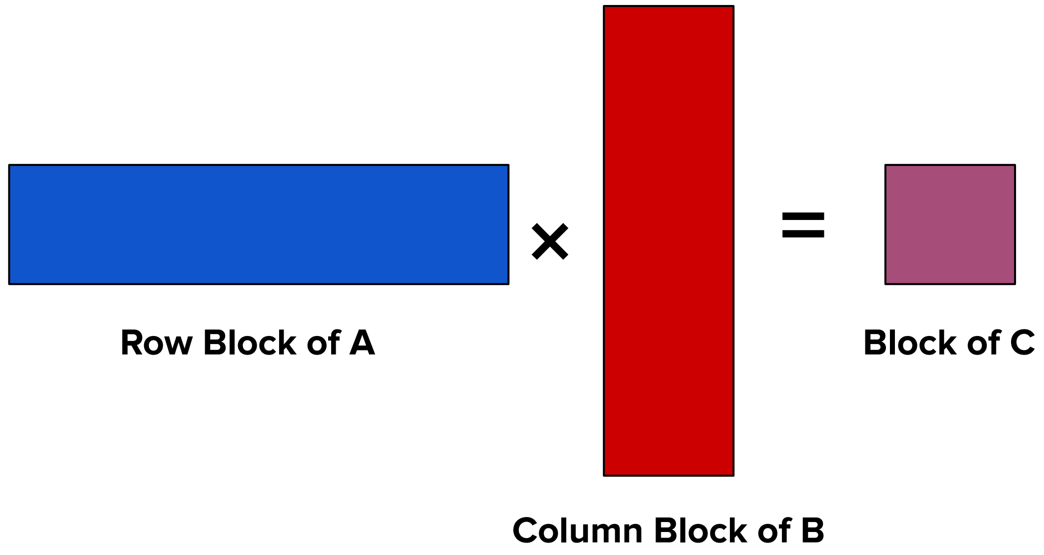
- Tiles are **assigned to processes** using some strategy

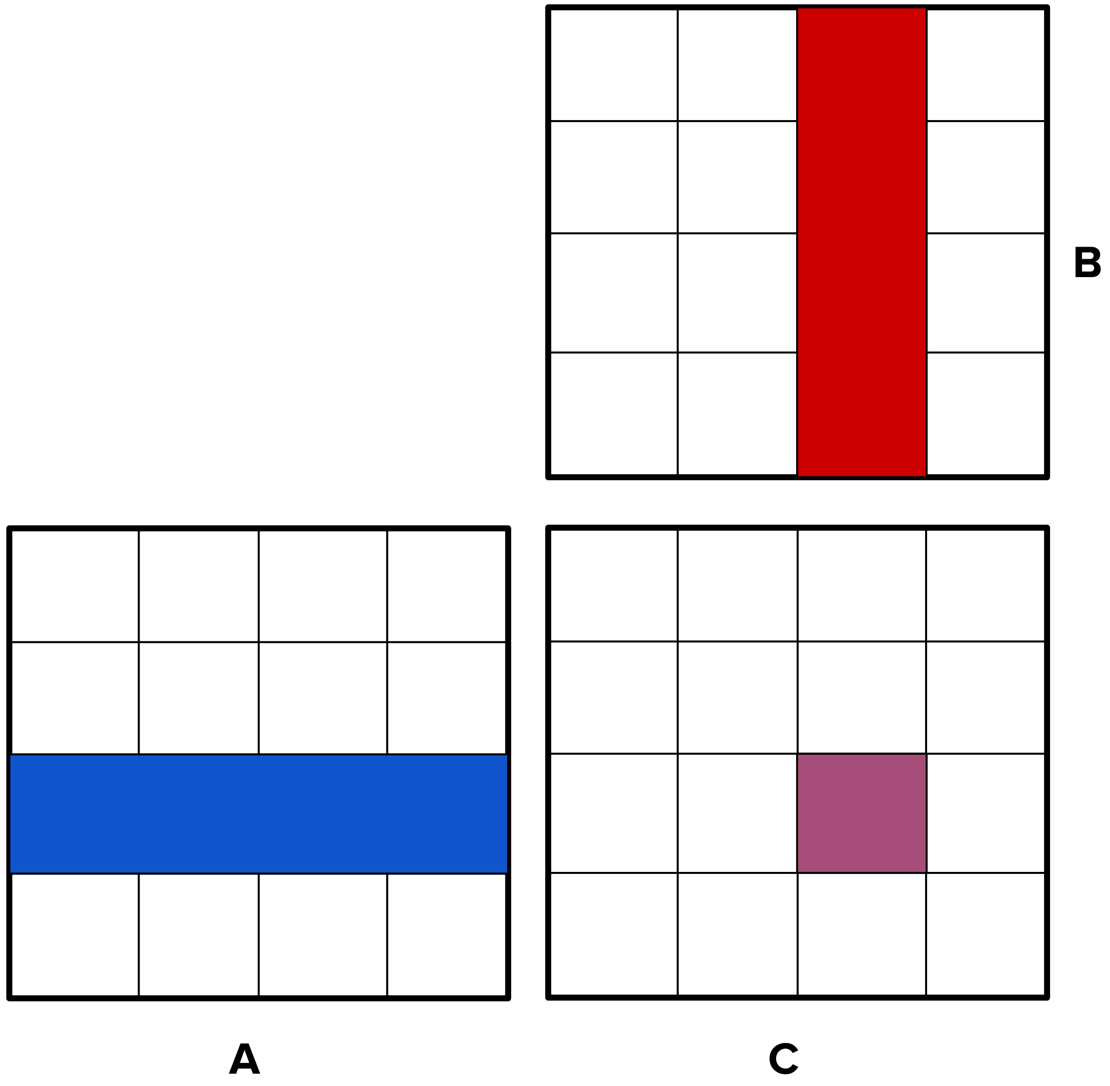


Distributed Matrix Multiplication Overview

We wish to compute **C** = **AB**

Let's compute **one block of the output, C**.

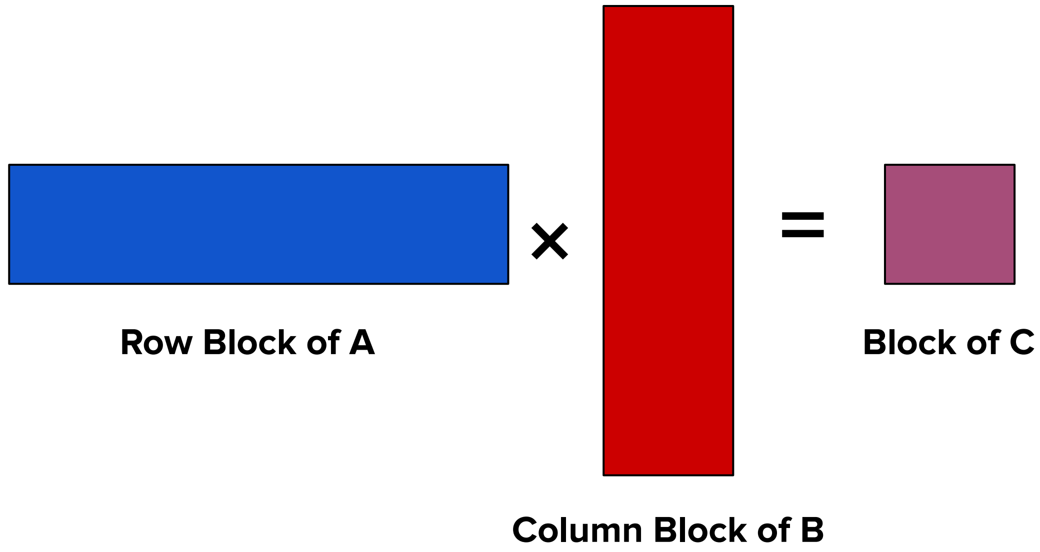


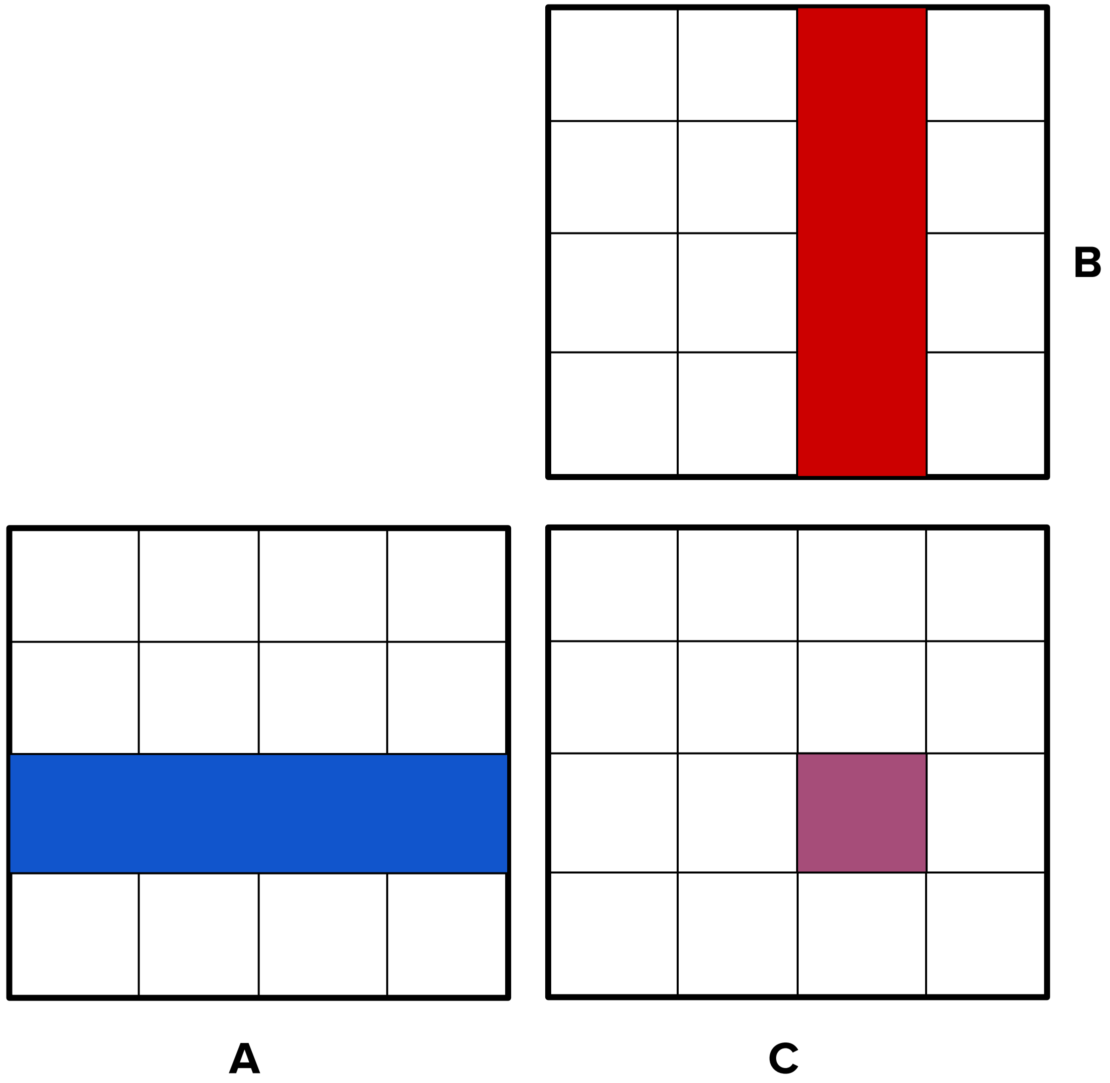


Distributed Matrix Multiplication Overview

We wish to compute **C = AB**

In practice, compute **one block at a time**



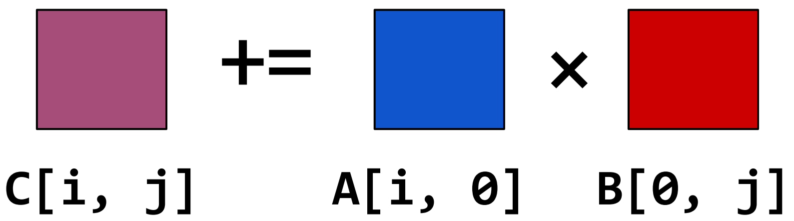


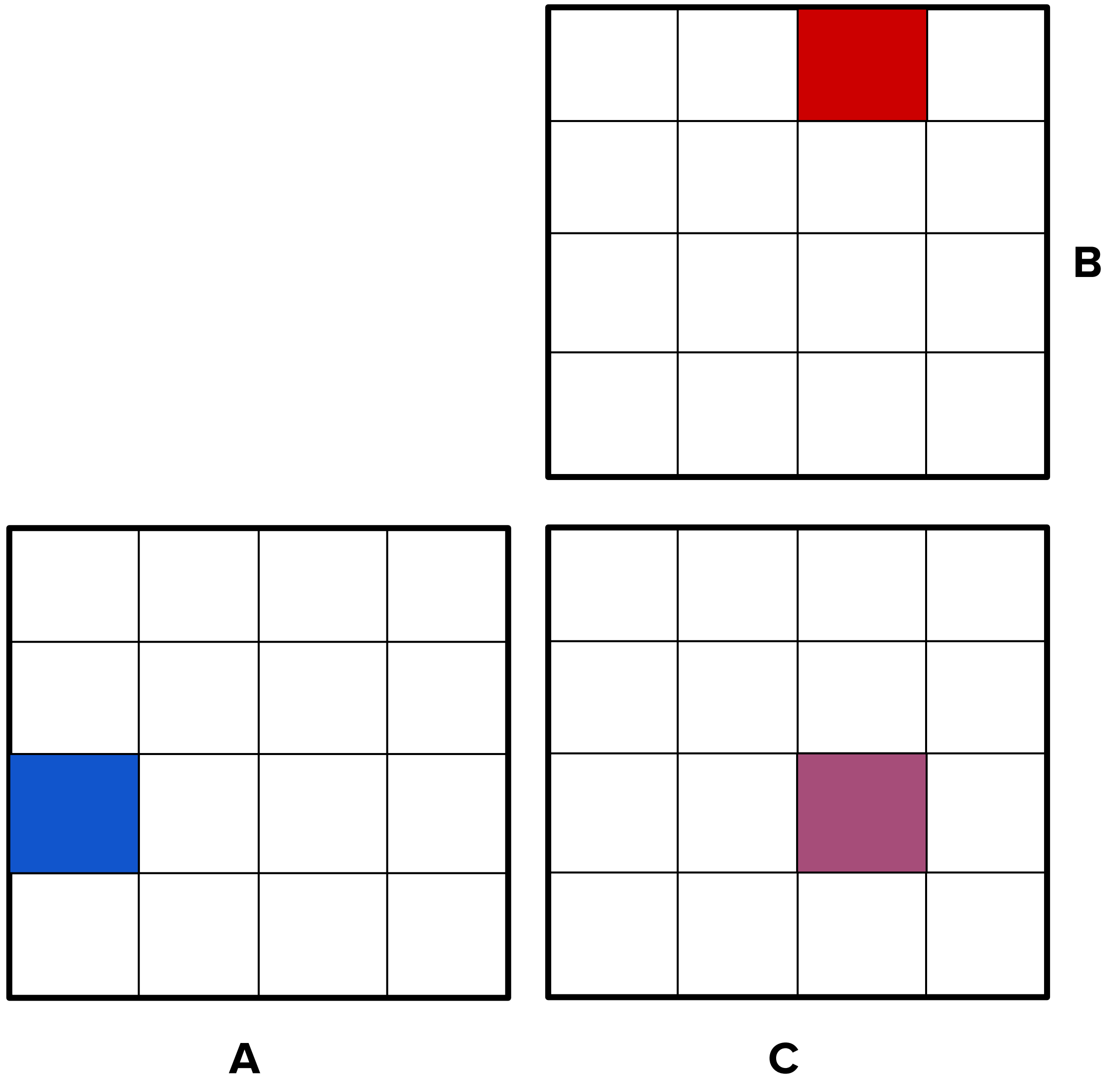
Distributed Matrix Multiplication Overview

We wish to compute **C = AB**

In practice, compute **one block at a time**

C[i, j] += A[i, k] * B[k, j] for all k



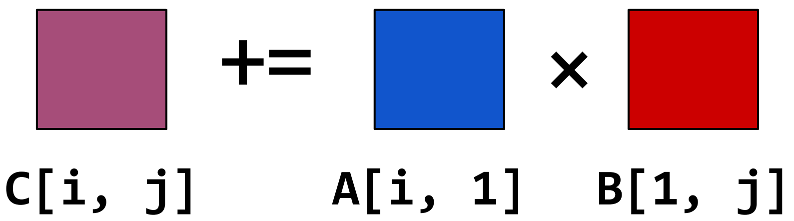


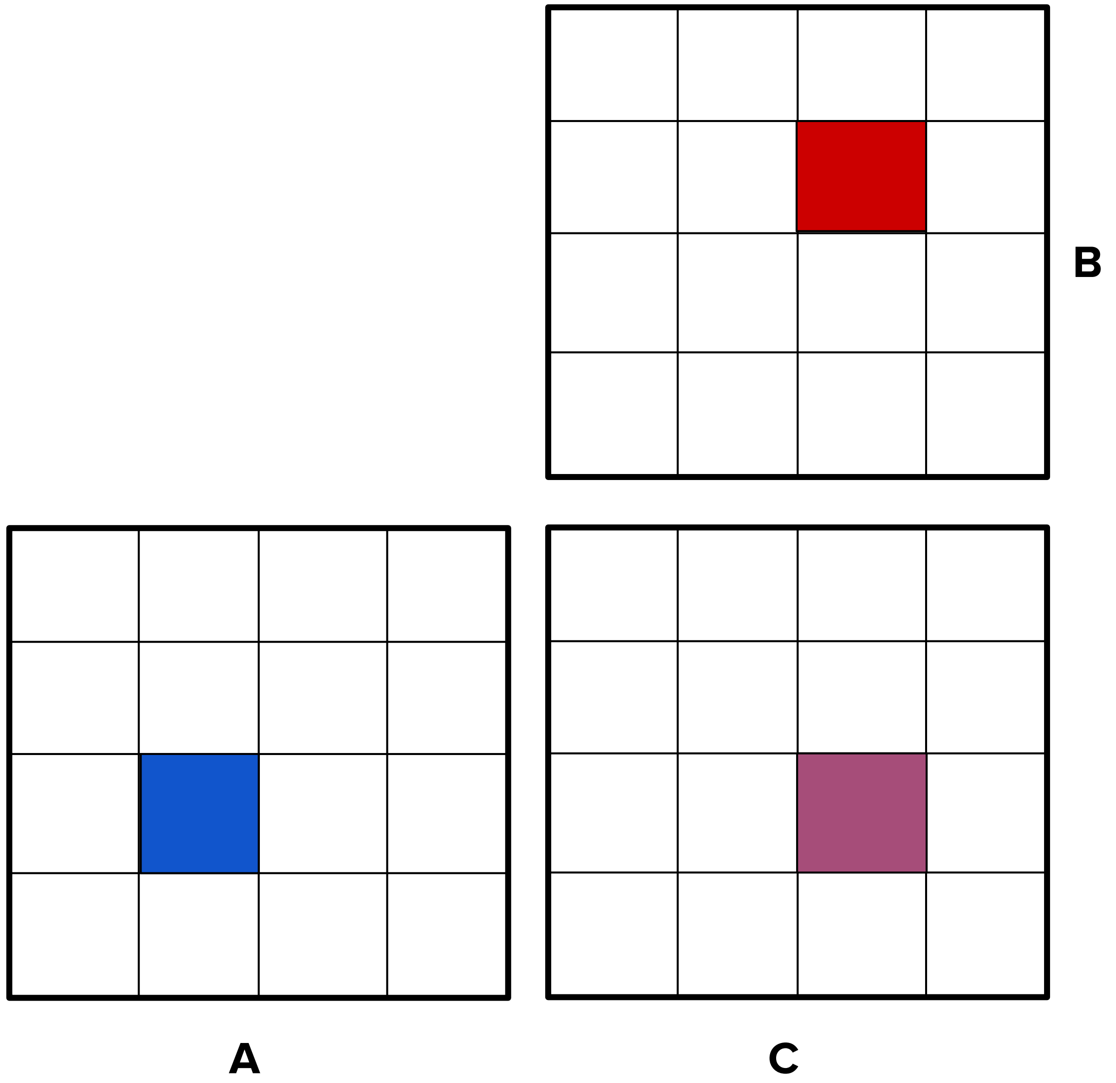
Distributed Matrix Multiplication Overview

We wish to compute **C = AB**

In practice, compute **one block at a time**

C[i, j] += A[i, k] * B[k, j] for all k



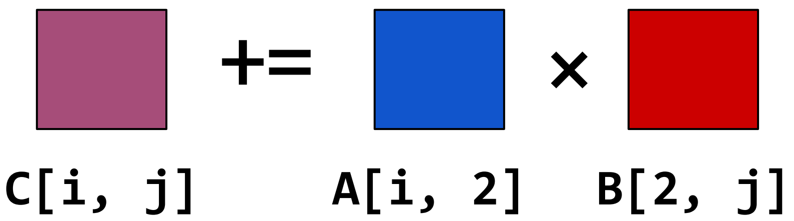


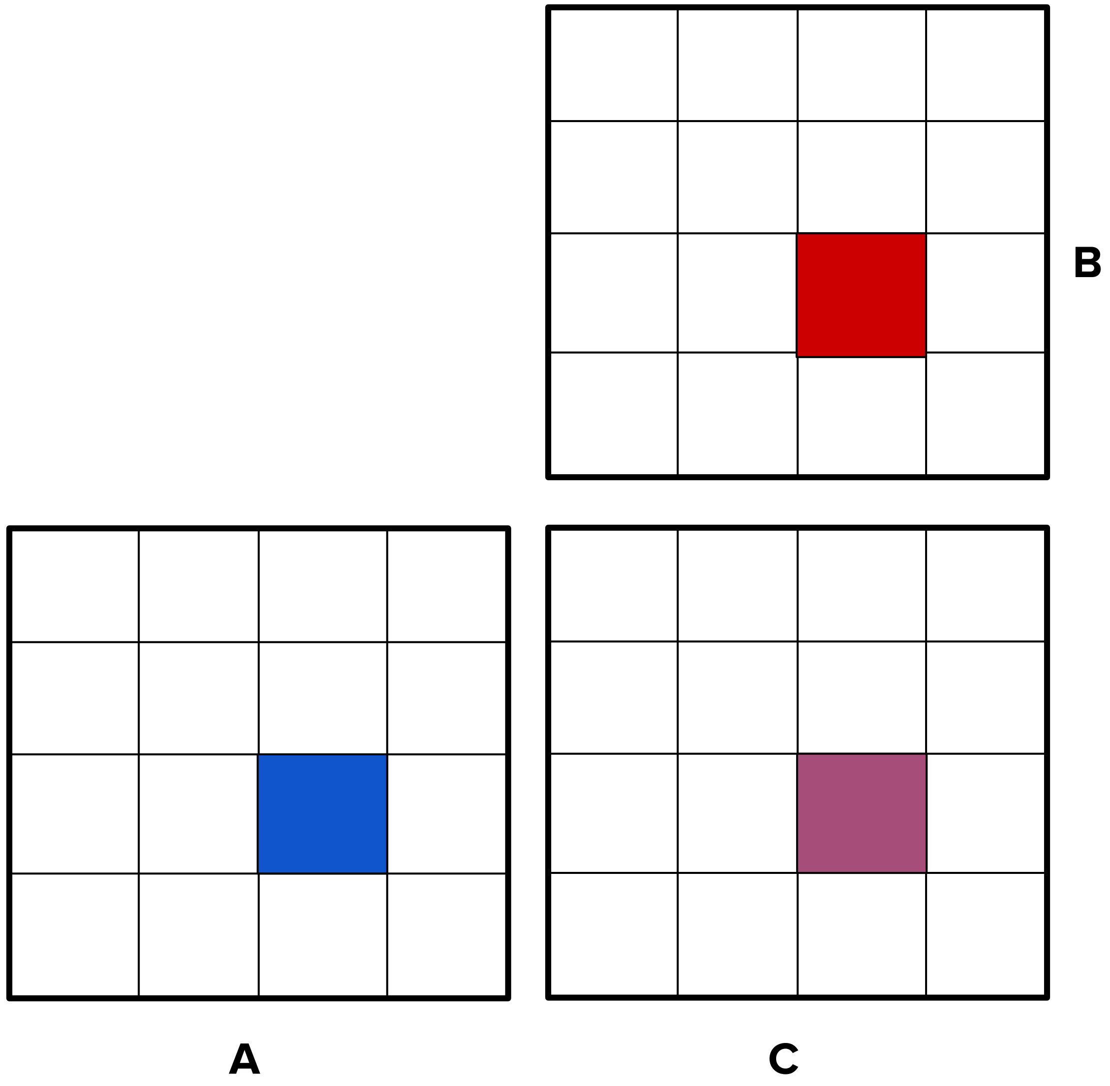
Distributed Matrix Multiplication Overview

We wish to compute **C = AB**

In practice, compute **one block at a time**

C[i, j] += A[i, k] * B[k, j] for all k



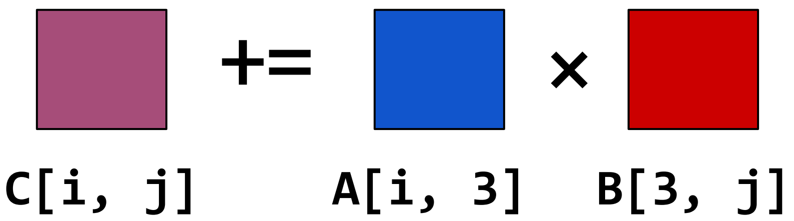


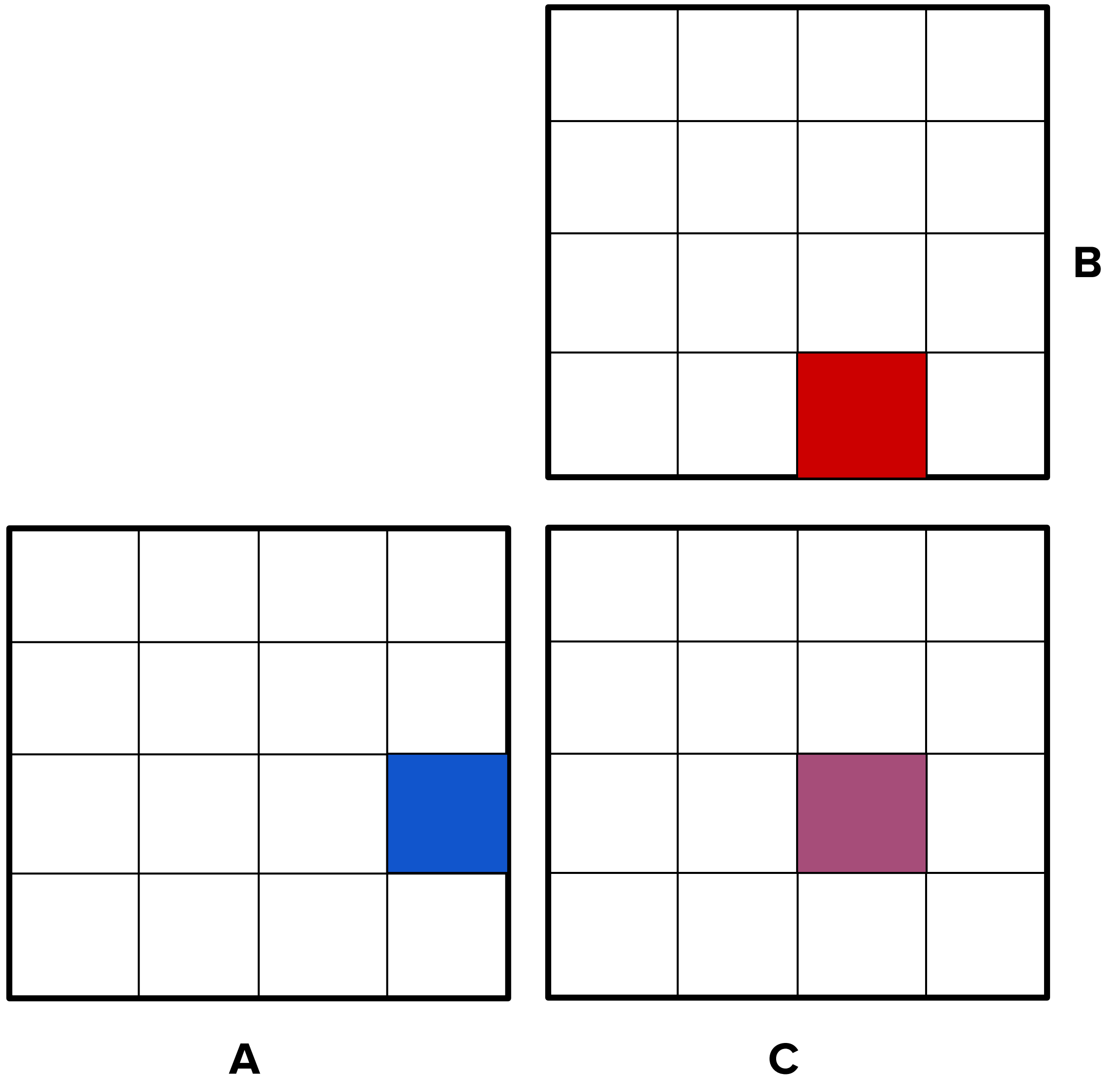
Distributed Matrix Multiplication Overview

We wish to compute **C = AB**

In practice, compute **one block at a time**

C[i, j] += A[i, k] * B[k, j] for all k

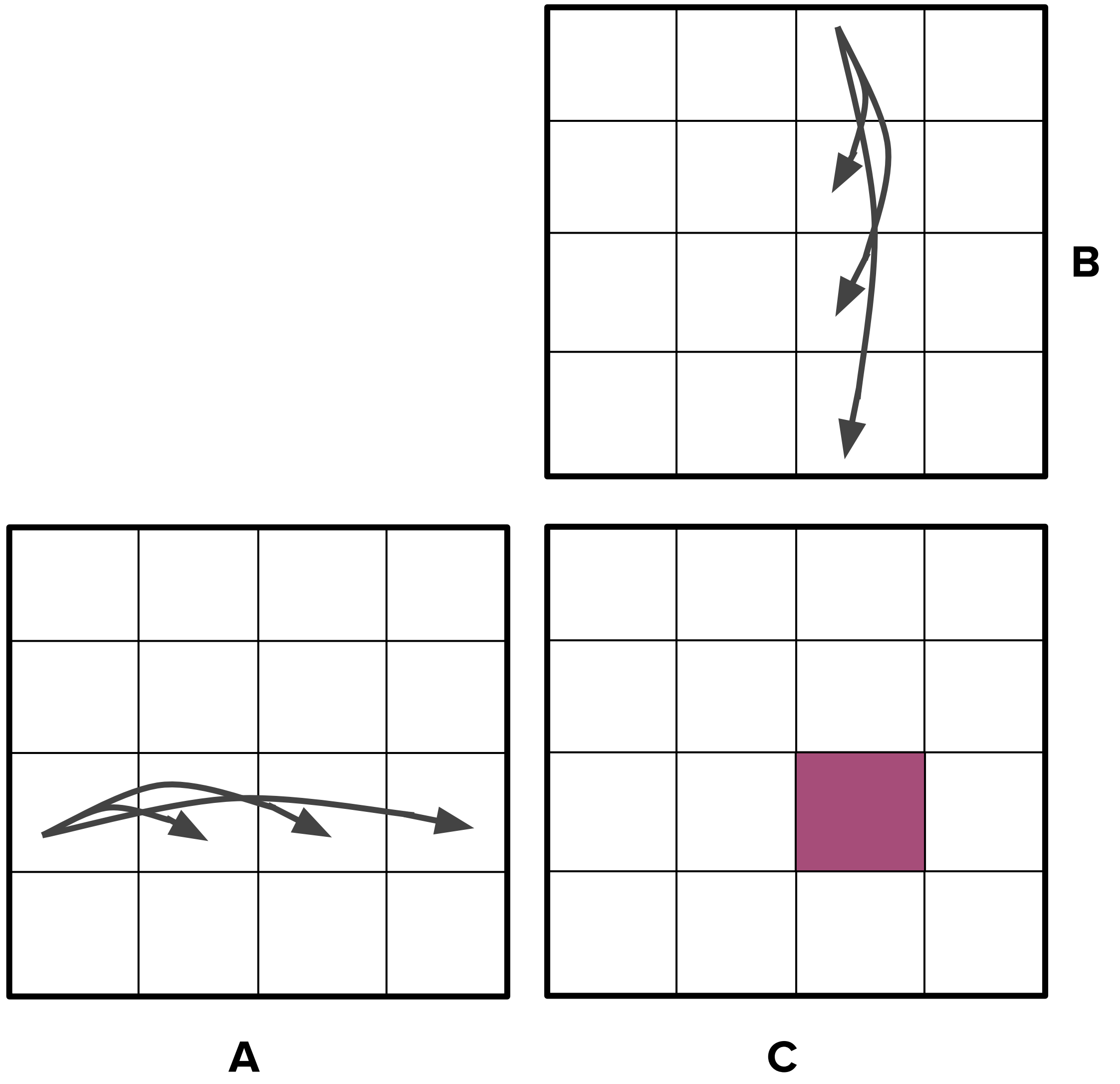




Methods of Moving Tiles

In **SUMMA**, row and column broadcasts distribute tiles

```
for k in K:
  broadcast in row of A -> local_a
  broadcast in column of B -> local_b
  local_c += local_a*local_b
```

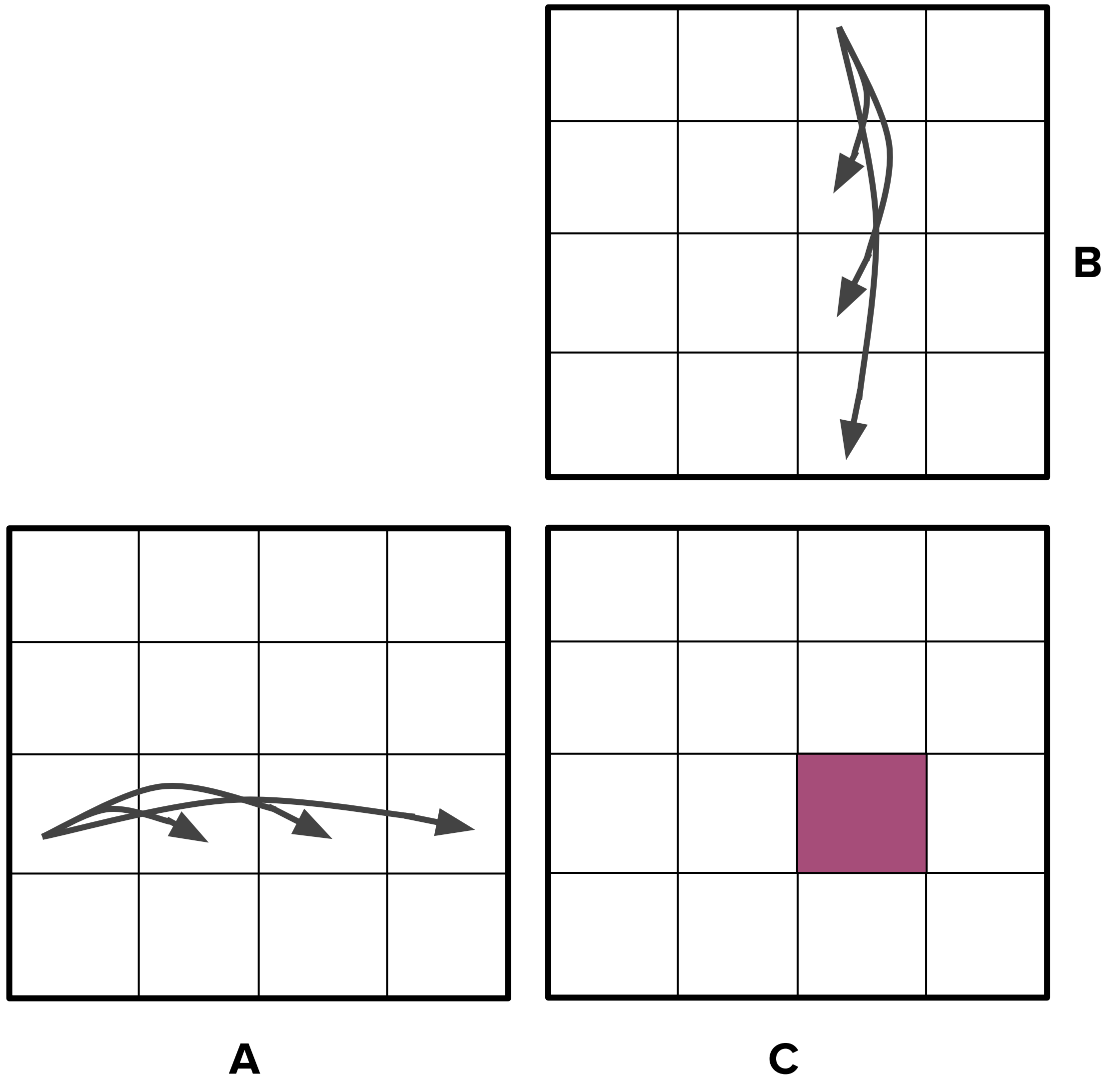


Methods of Moving Tiles

In **SUMMA**, row and column broadcasts distribute tiles

```
for k in K:
  broadcast in row of A -> local_a
  broadcast in column of B -> local_b
  local_c += local_a*local_b
```

Explicit barrier!

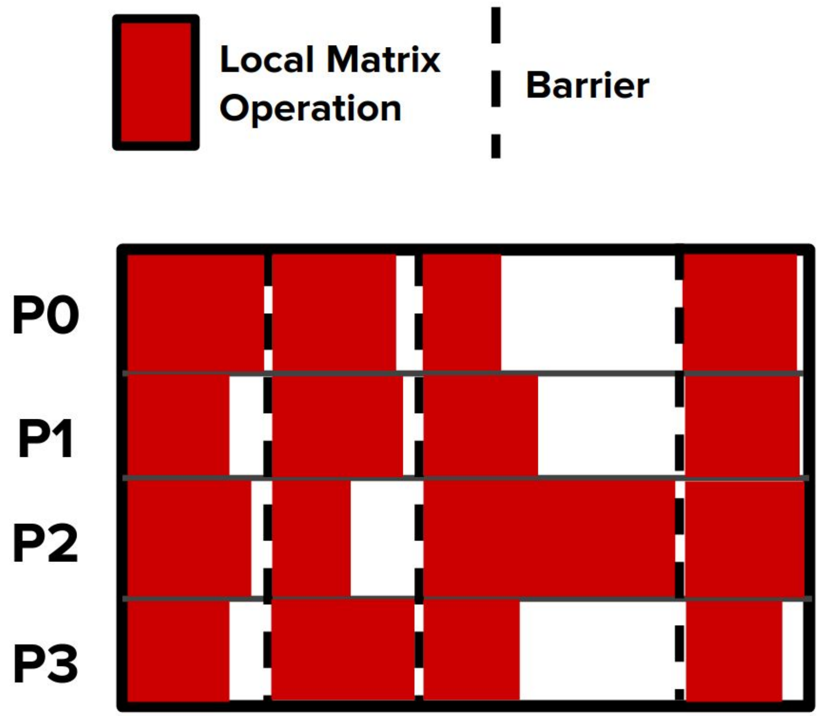


An Issue with Bulk Synchronous Distributed MM

- In **bulk synchronous algorithms**, load balancing problems can occur

- With **local sparse operations, each operation** may have differing amounts of work

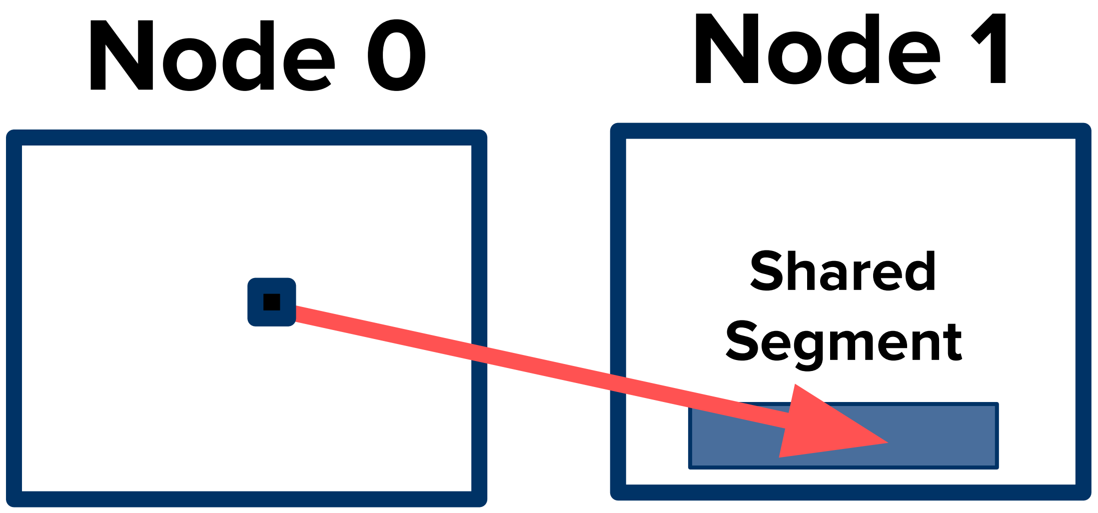
- This leads to **time wasted** waiting for slower processes to finish



Hypothetical Execution Timeline

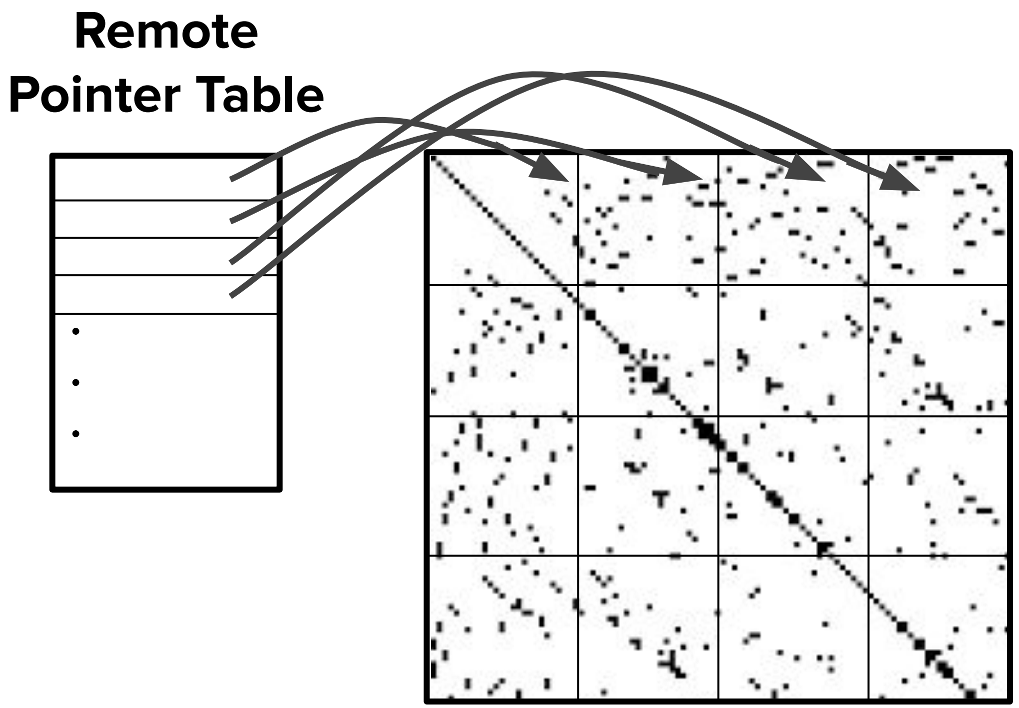
RDMA-Based Algorithms

- RDMA provides **put** and **get operations**
- **Put** writes to a remote node's memory, **get** reads
- We can use **RDMA** to implement **distributed matmul**



RDMA-Based Matrix Data Structure

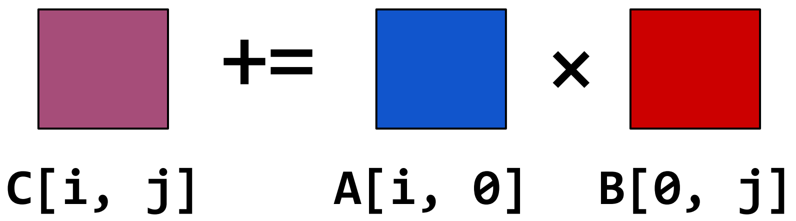
- Each process has a **remote pointer** it can use to **get / put** to a tile
- In the **dense matrix** case, single pointer
- In the **sparse case**, pointers to **CSR data structure**

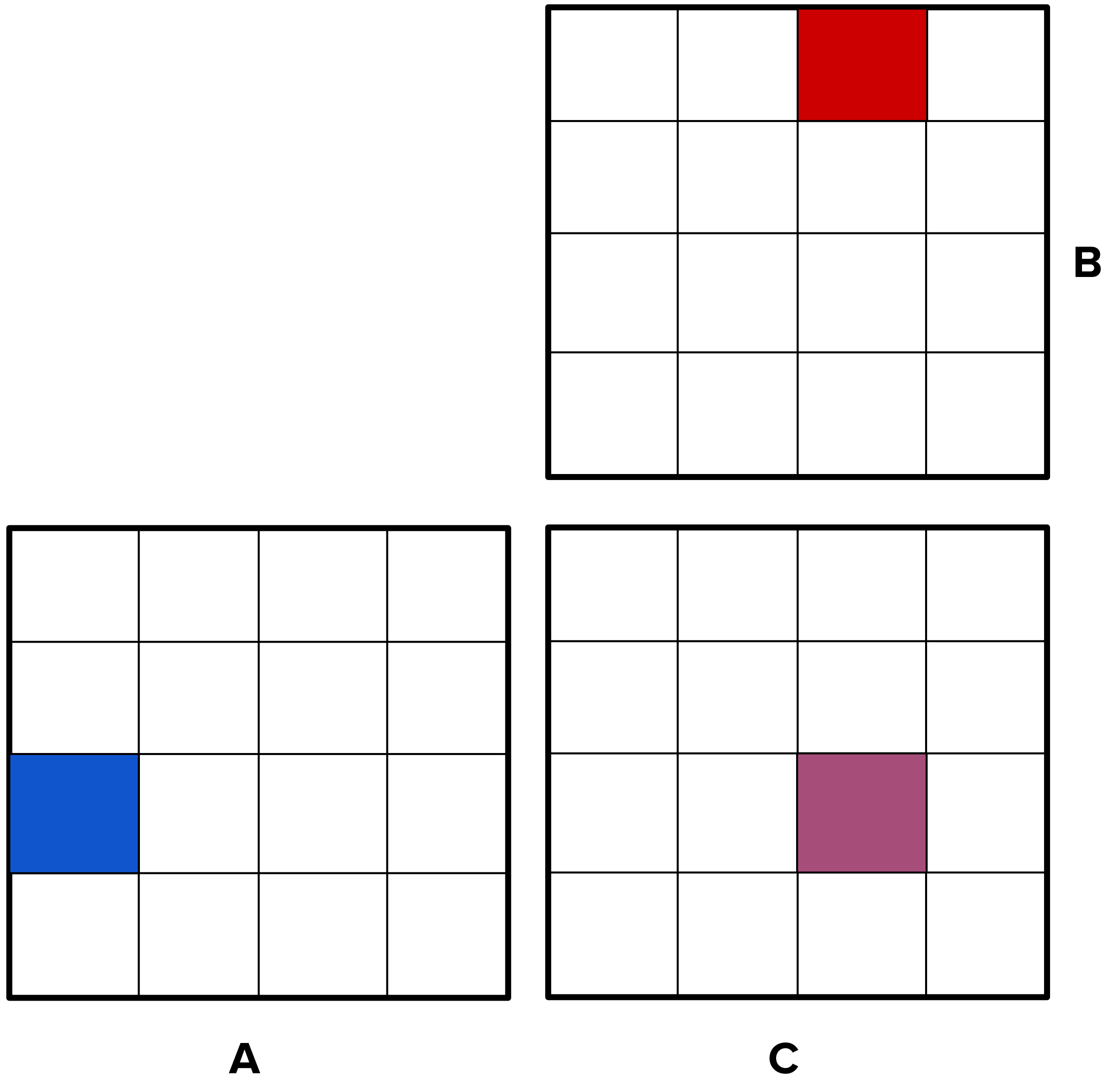


Distributed Matrix Multiplication Overview

We wish to compute **C** = **AB**

```
i, j = my_block(C)
for k in K:
  local_a = A[i, k].get()
  local_b = B[k, j].get()
  local_c += local_a*local_b
```

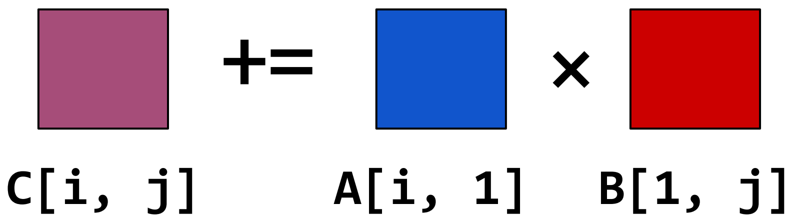


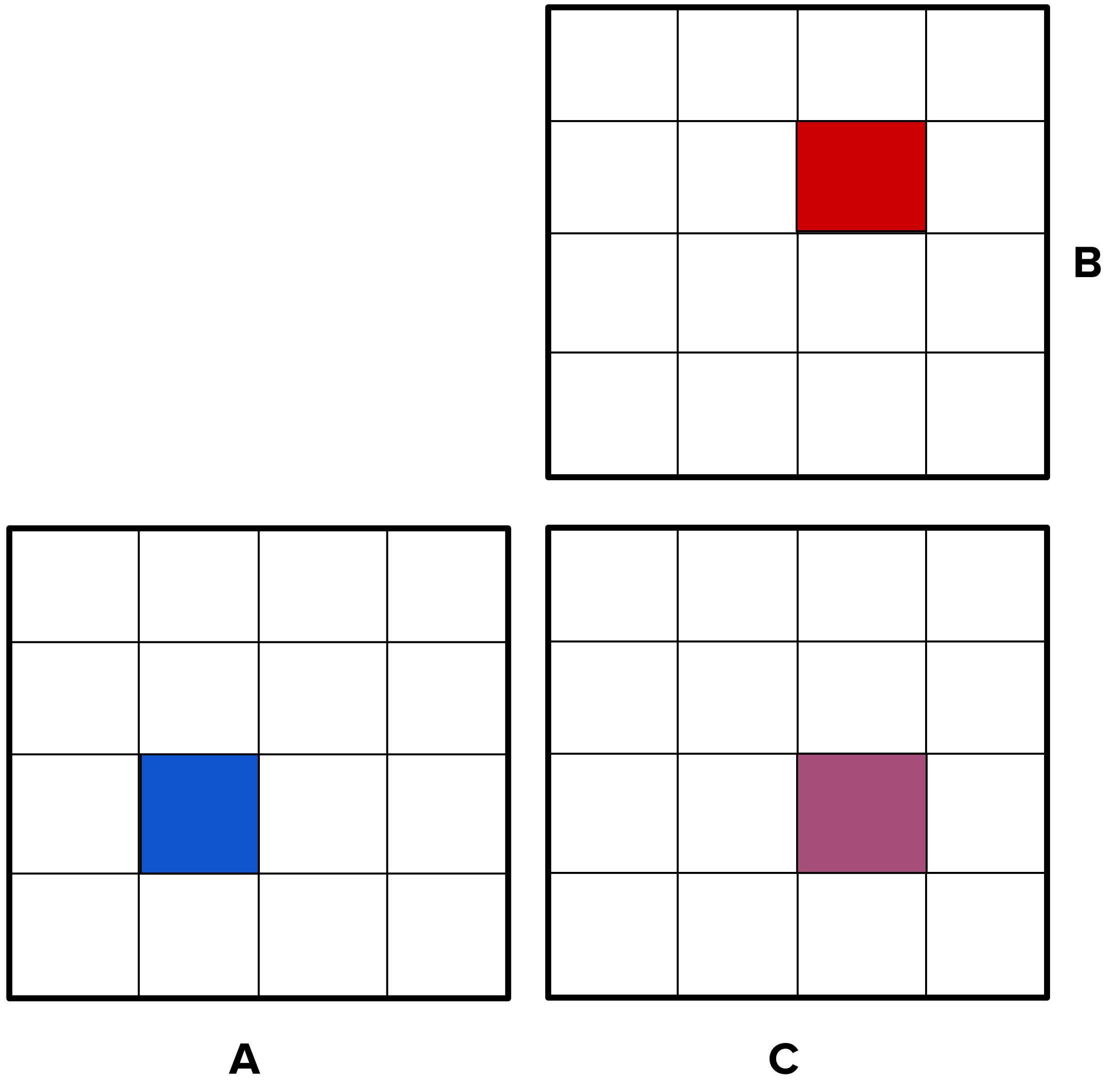


Distributed Matrix Multiplication Overview

We wish to compute **C = AB**

```
i, j = my_block(C)
for k in K:
  local_a = A[i, k].get()
  local_b = B[k, j].get()
  local_c += local_a*local_b
```

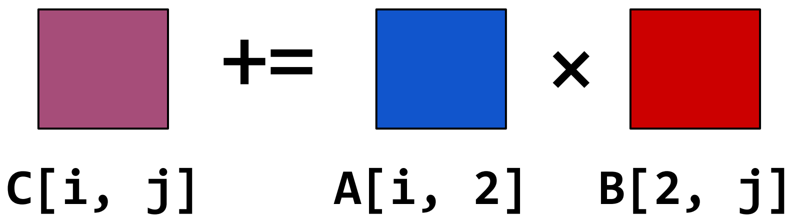


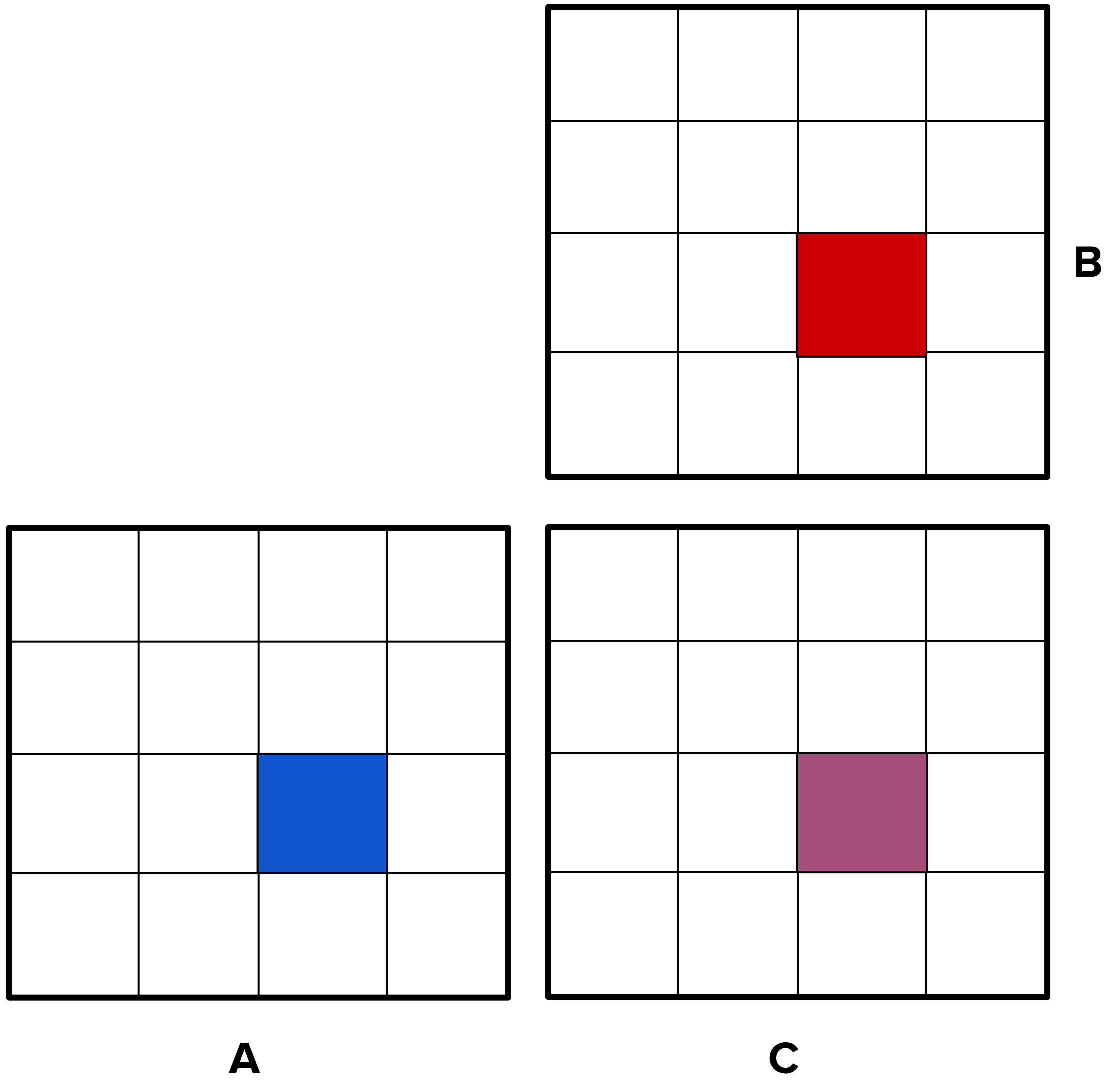


Distributed Matrix Multiplication Overview

We wish to compute **C** = **AB**

```
i, j = my_block(C)
for k in K:
  local_a = A[i, k].get()
  local_b = B[k, j].get()
  local_c += local_a*local_b
```

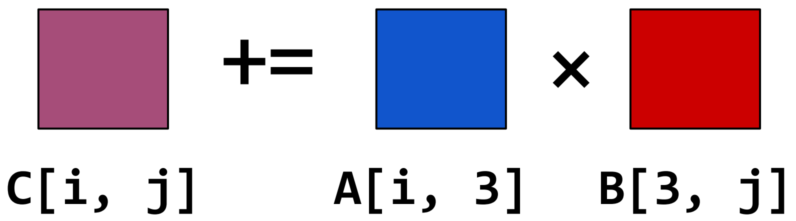


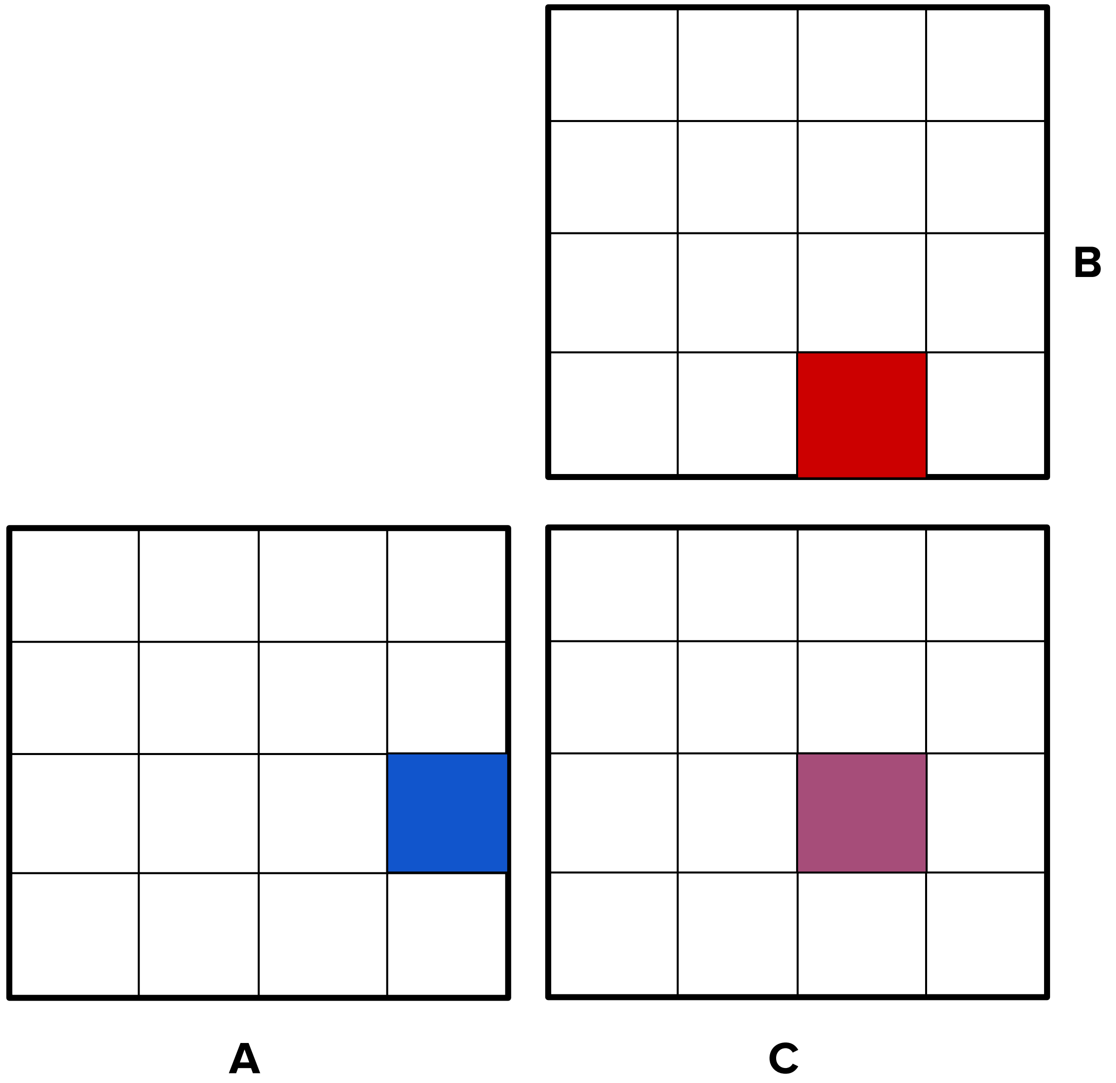


Distributed Matrix Multiplication Overview

We wish to compute **C** = **AB**

```
i, j = my_block(C)
for k in K:
  local_a = A[i, k].get()
  local_b = B[k, j].get()
  local_c += local_a*local_b
```



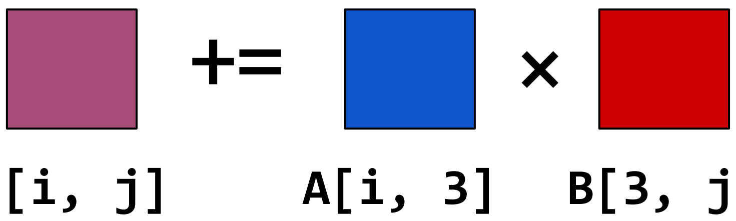


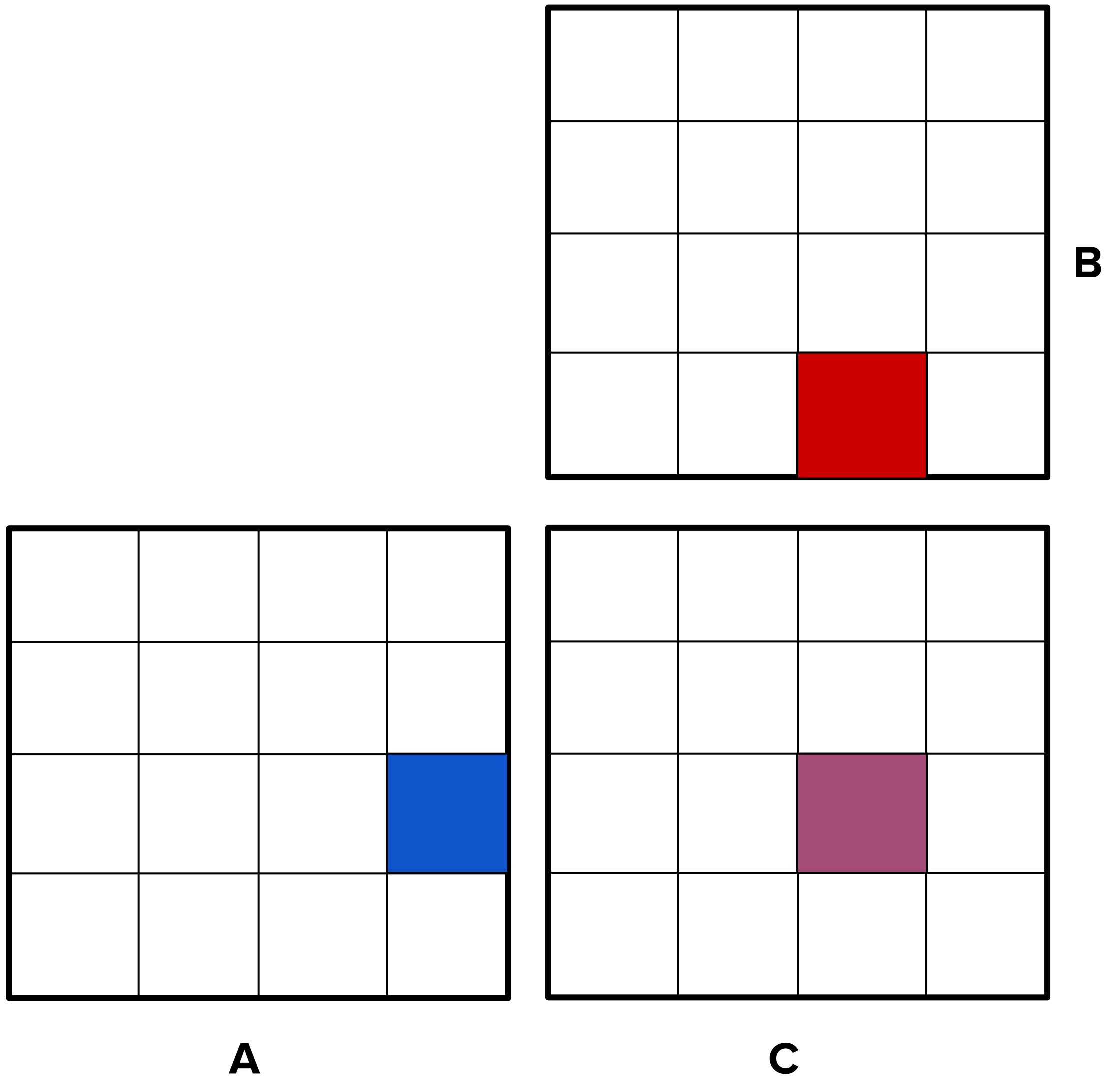
Distributed Matrix Multiplication Overview

We wish to compute **C** = **AB**

```
i, j = my_block(C)
for k in K:
  local_a = A[i, k].get()
  local_b = B[k, j].get()
  local_c += local_a*local_b
```

No Barrier!



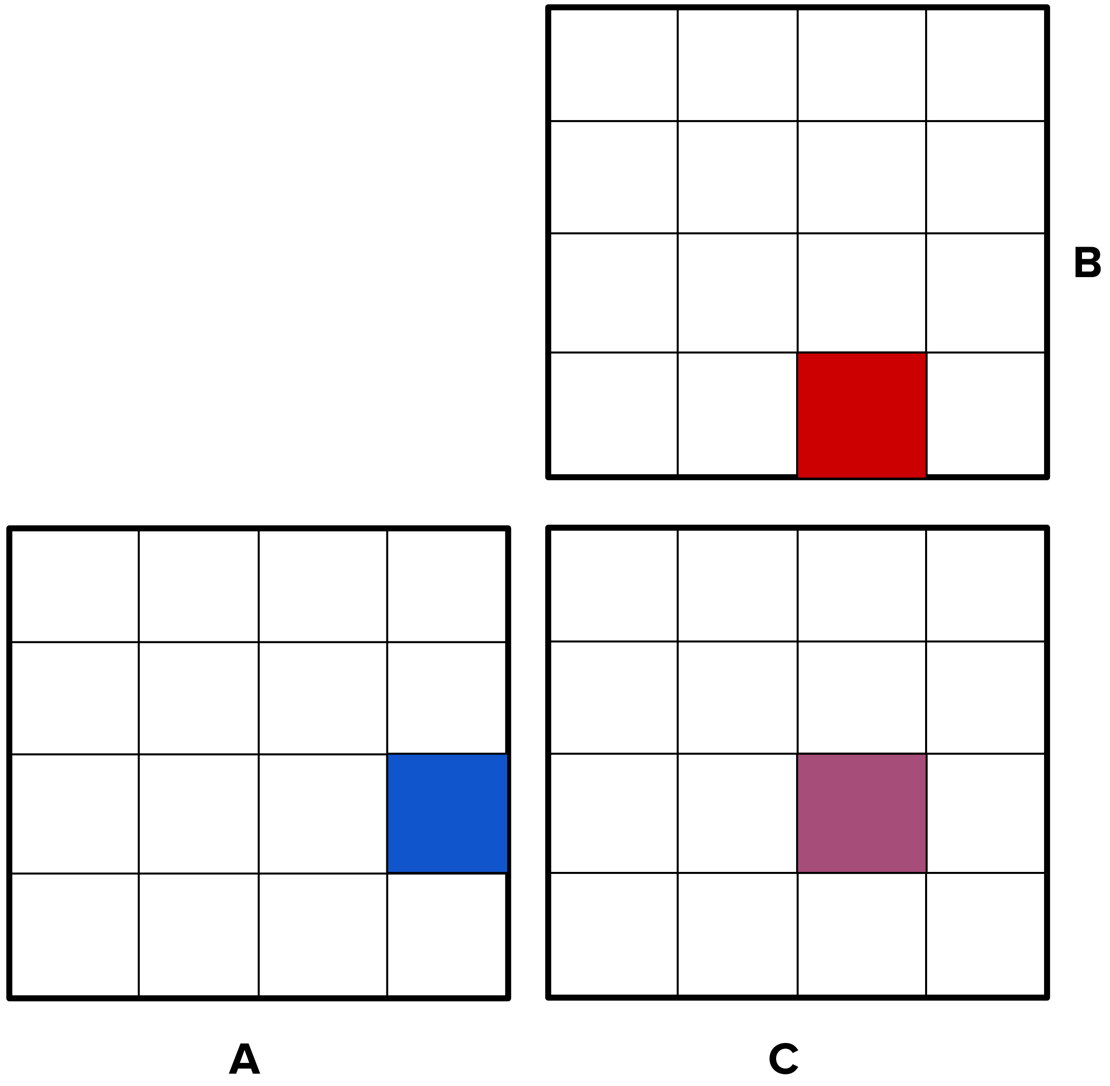


Important Optimizations

We wish to compute **C** = **AB**

```
i, j = my_block(C)
for k_ in K:
  k = (k_ + k_offset) % K
  local_a = A[i, k].get()
  local_b = B[k, j].get()
  local_c += local_a*local_b
```

1) Iteration offset



Important Optimizations

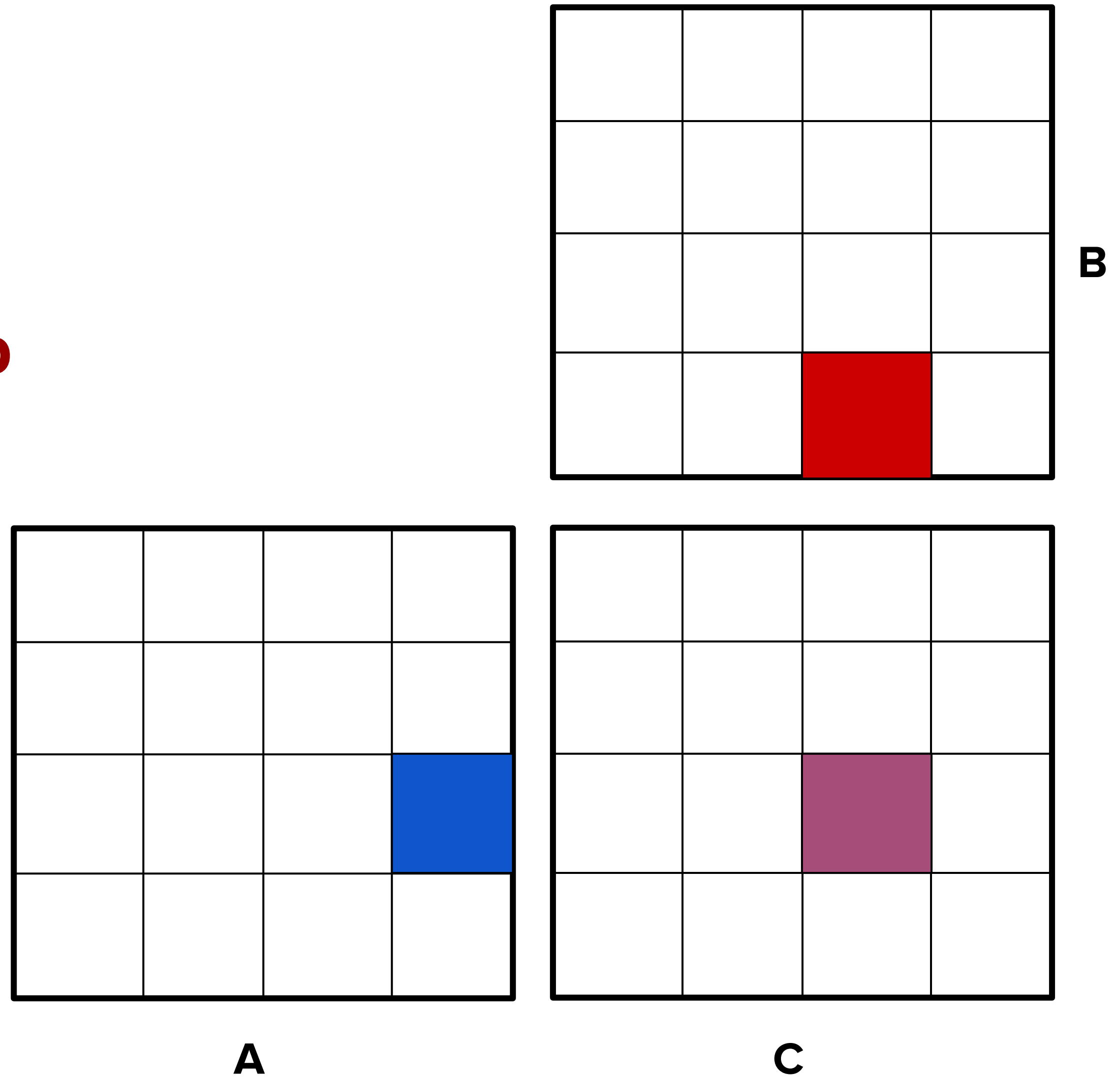
We wish to compute **C** = **AB**

```
i, j = my_block(C)
for k_ in K:
  k = (k_ + k_offset) % K
  local_a = buf_a.get()
  local_b = buf_b.get()
  if k_ + 1 < K:
    buf_a = A[i, k+1].async_get()
    buf_b = B[k+1, j].async_get()

  local_c += local_a*local_b
```

1) Iteration offset

2) Pre-fetching, for overlap



Stationary A, B, and C Implementations

- It should be noted that thus far, we’ve implied a **stationary C** implementation

- With **stationary C**, **C remains in place**, while A and B must be **communicated**

- We’ve also implemented **RDMA stationary A&B**

Performance Results

Performance Implementations

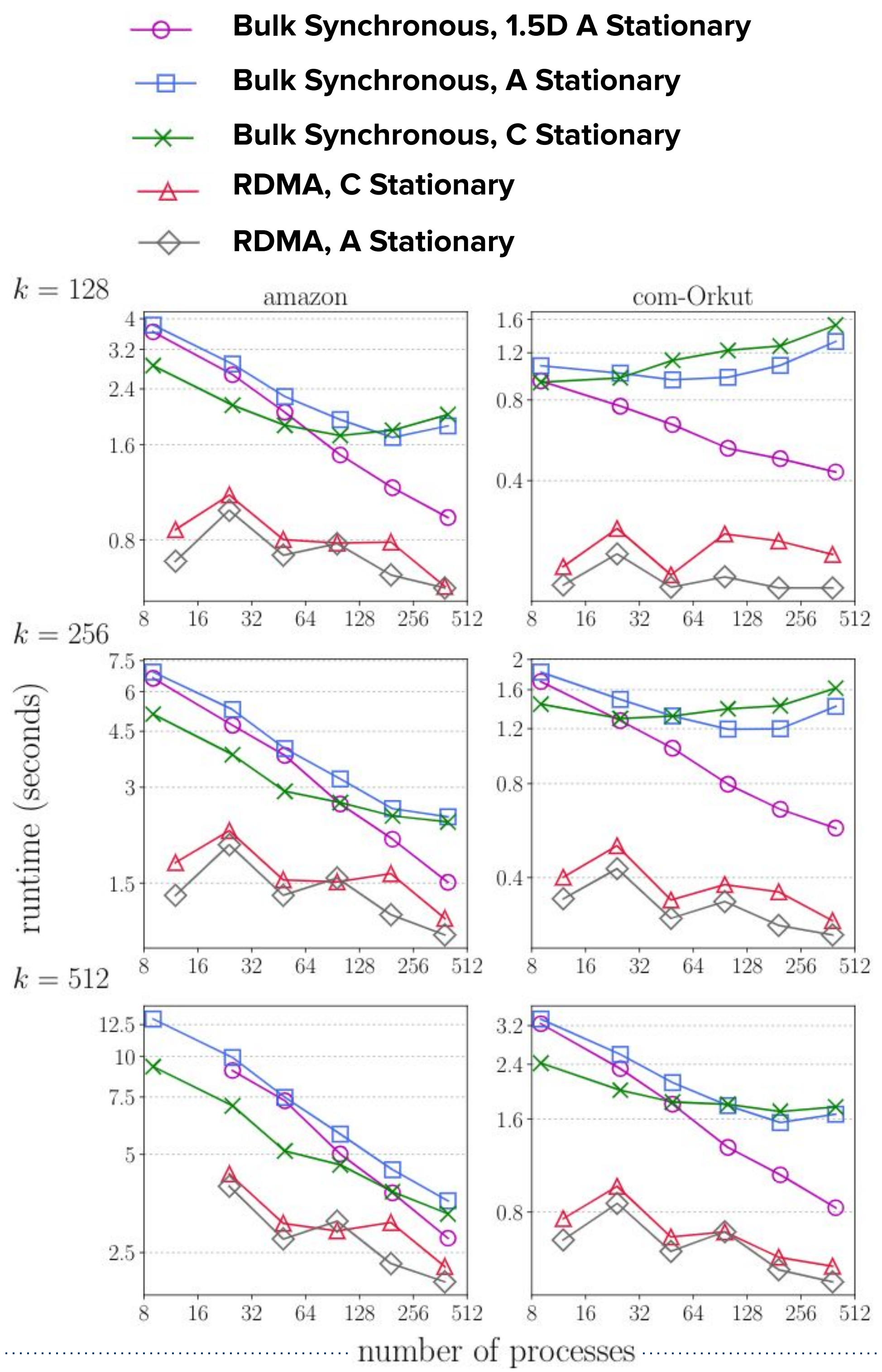
- We implemented **dense and sparse matrix data structures** on using BCL, for both distributed **CPU** and **GPU**

- Results presented today are for SpMM GPU, using **NVSHMEM**, an **extension of OpenSHMEM** that provides direct GPU-to-GPU communication

- **cuSPARSE** used for local sparse matrix operations

SpMM (Sparse times Dense)

- **Bulk synchronous implementations** (top 3 lines) use **CUDA-aware MPI**
- **Asynchronous implementations** (bottom 2 lines) use **NVSHMEM**
- **All implementations** use **CuSPARSE** for local computation.



*All experiments run on OLCF's Summit. y-axis is runtime, x-axis is number of Tesla V100 GPUs.

Berkeley
UNIVERSITY OF CALIFORNIA

Conclusions

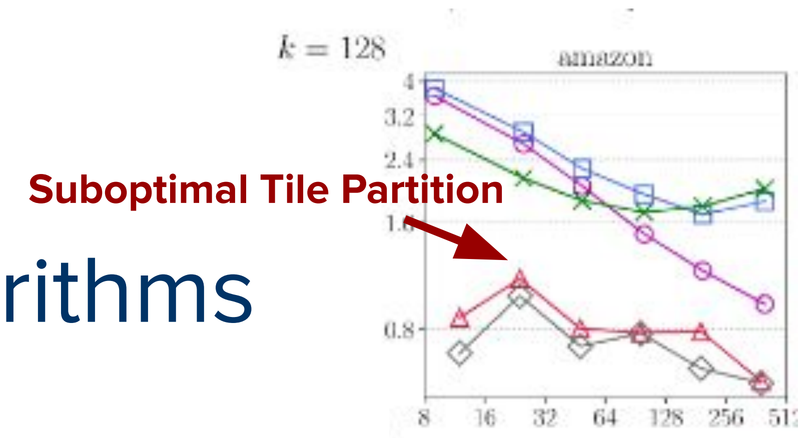
1. **RDMA-based implementations** of distributed matrix multiply **decouple** inner loop iterations and are truly asynchronous

2. They **perform favorably** compared to bulk synchronous implementations

3. As with many sparse operations, can be **difficult to scale** if not enough work

Limitations

1. Many graph algorithms require **custom semirings**
   a. **CuSPARSE** does not currently support custom semirings
   b. Currently evaluating **GE-SpMM** and **CUSP**



2. Still experimenting with **tile partitioning** algorithms

3. RDMA-based Stationary A,B algorithms can be **less memory efficient** than bulk synchronous implementations

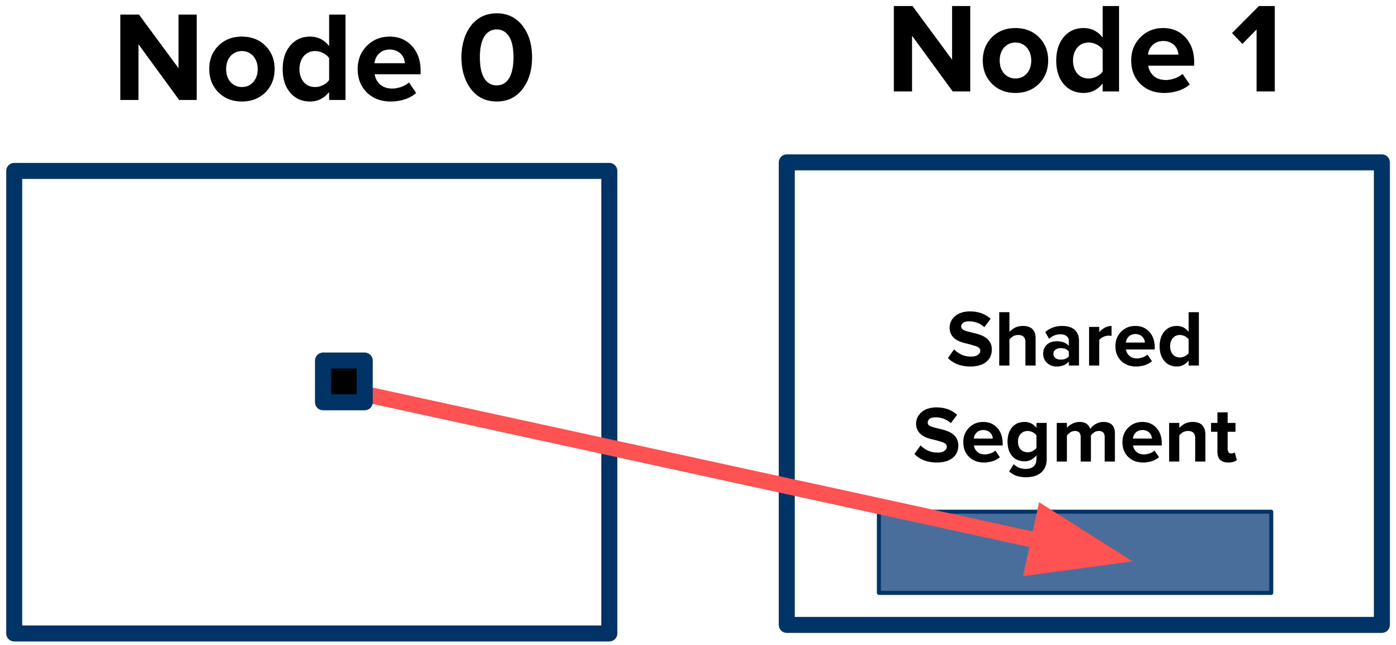
Backup Slides

Berkeley Container Library

- A series of data structures built on **global pointers**

- Processes can directly **read and write** from each others' memories

- Executed in **RDMA**



Berkeley Container Library Philosophy

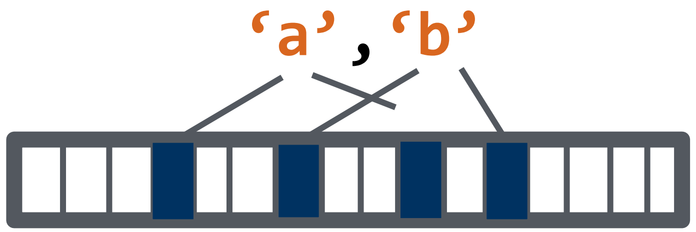
- Use **RDMA** for all principal data structure operations

1) Executed efficiently in **hardware**

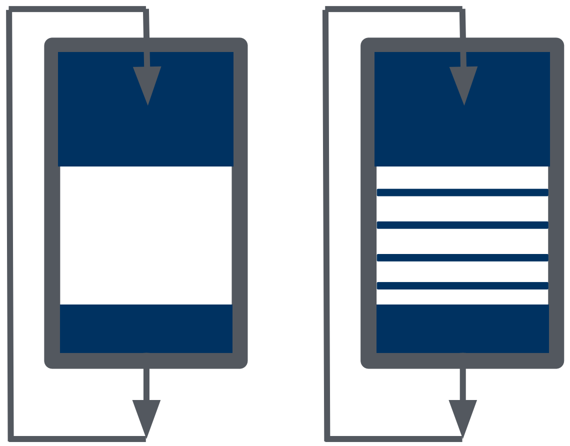
2) No need to **interrupt** remote CPU

3) **Maps well** to familiar data structure operations

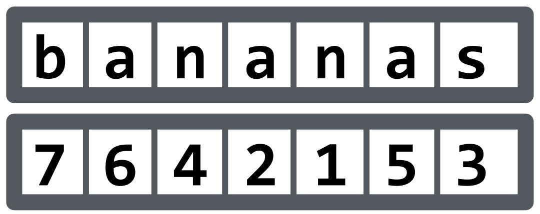
BCL Data Structures



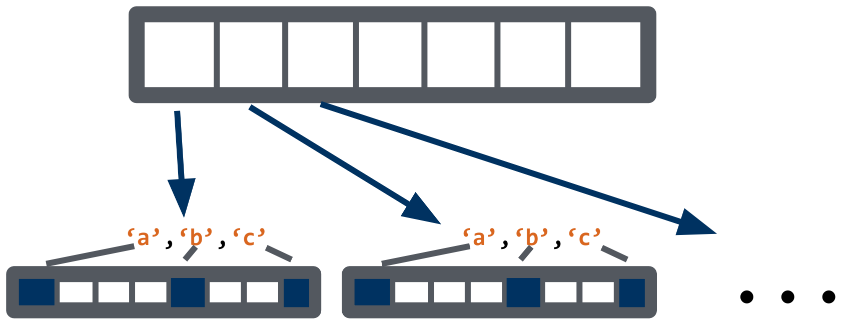
Bloom filters



Queues



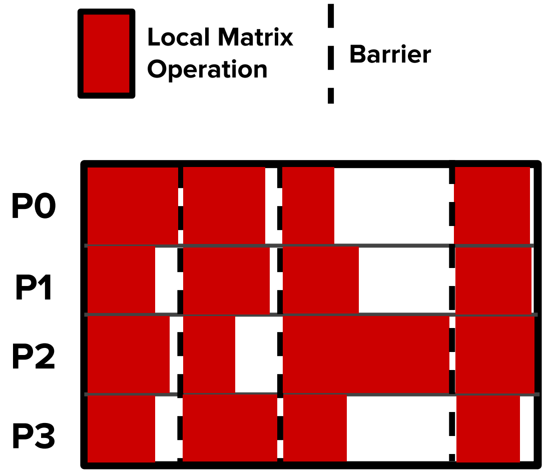
Suffix arrays



Hash tables

Drawings / Content

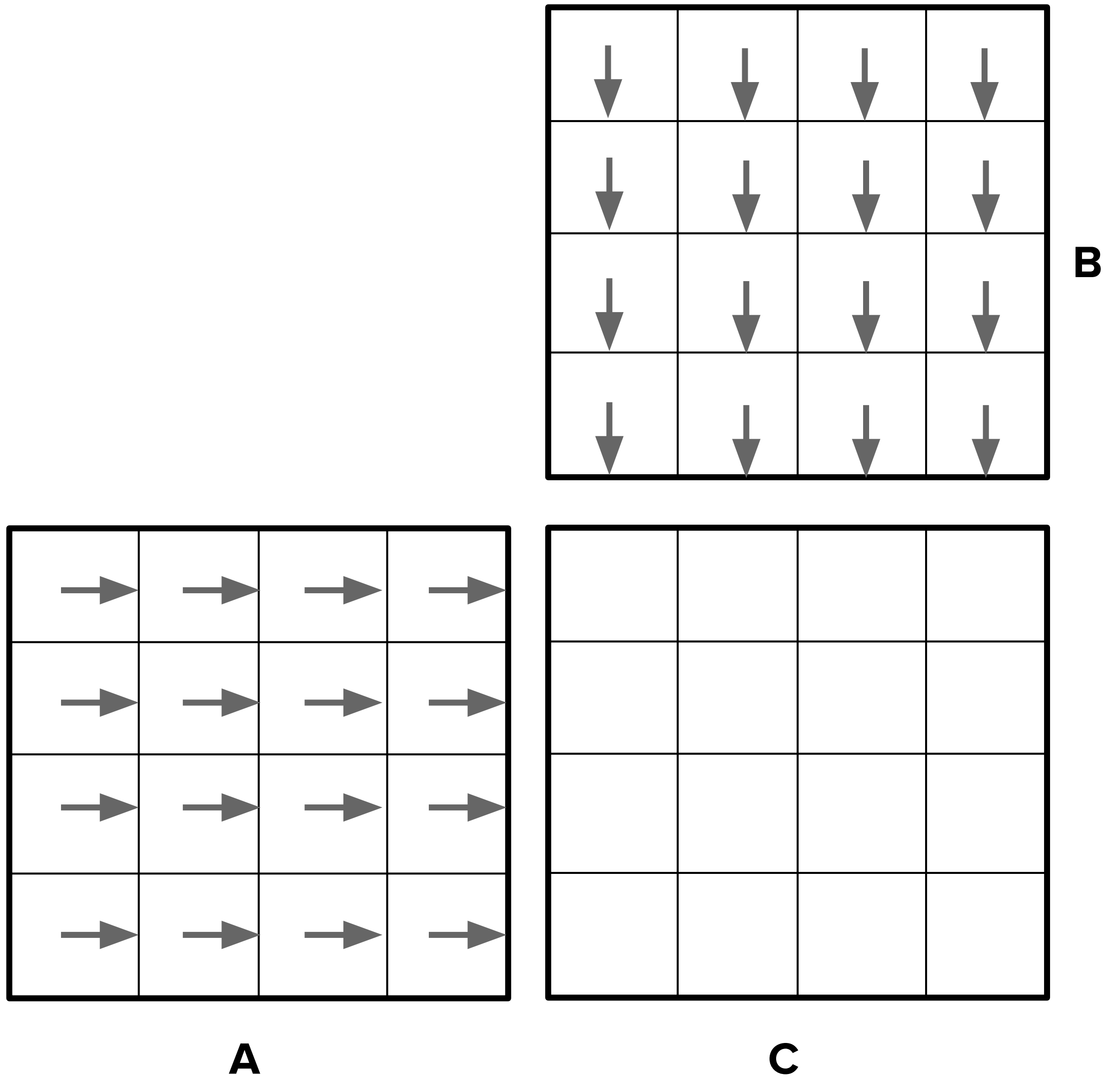
An Issue with Bulk Synchronous Distributed MM



Methods of Moving Tiles

In **Cannon's algorithm**, a **redistribution step**, followed by **passing matrices** right and below

```
for k in K:
   send A tile to the right
   send B tile below
   receive A, receive B
   local_c += A*B
```



Methods of Moving Tiles

In **Cannon's algorithm**, a **redistribution step**, followed by **passing matrices** right and below

```
for k in K:
   send A tile to the right
   send B tile below
   receive A, receive B
   local_c += A*B
```

Implicit barrier!